# Activity Recognition with Intended Actions

**Alfredo Gabaldon**
Center for Artificial Intelligence
New University of Lisbon
ag@di.fct.unl.pt

## Abstract

The following activity recognition problem is considered: a description of the action capabilities of an agent being observed is given. This includes the preconditions and effects of atomic actions and of the activities (sequences of actions) the agent may execute. Given this description and a set of propositions, called *history*, about action occurrences, intended actions and properties of the world all at various points in time, the problem is to complete the picture as much as possible and determine what has already happened, what the intentions of the agent are, and what may happen as a result of the agent acting on those intentions. We present a framework to solve these activity recognition problems based on a formal language for reasoning about actions that includes a notion of intended actions, and a corresponding formalization in answer set programming.

## 1 Introduction

We consider the following problem: given a partial record of what an agent being observed is doing, including a) intended actions, b) action executions, c) fluent values, all at various time points, determine a complete picture of what the agent has done, intends and may do in the future. This *activity recognition* problem is what we are concerned with in this paper. We develop a new approach that is based on logical reasoning with partial information about the activities of an observed agent and a background knowledge base that includes a formal action theory representing how the world evolves as the agent executes actions, knowledge about non-elementary actions, called *activities*, that the agent might be executing, and a theory of intended actions.

For illustration of our approach we shall use the following adapted version of an example from [Kautz and Allen, 1986].

**Example 1** The observed agent in this domain is capable of executing various meal preparation activities: it can *cook chicken marinara* which consists in *making marinara* sauce and putting it together with chicken by *mixing chicken marinara*. It can *cook fettuccini alfredo* by *making fettuccini*, *making alfredo* sauce and putting it together by *mixing fettuccini alfredo*. It can also *cook fettuccini marinara*, *cook spaghetti carbonara* and *cook chicken primavera* using similar steps. Then, if the agent declares that he intends to *make fettuccini* at time 1, and we observe that *mix chicken marinara* occurred at time 3, then the possible conclusions are that the agent intends to cook two dishes: one of the fettuccini dishes and chicken marinara. In the case that he is making fettuccini marinara, it is possible that he makes marinara sauce once, for both dishes, i.e. the two cooking activities share an action.

There is a substantial body of work on activity or plan recognition. Among the influential early work on this problem is [Kautz and Allen, 1986], which presents a formal framework for plan recognition using logic and circumscription. Our approach is also logic-based and aims at a fairly general solution. However, we build on a more recent and general framework for reasoning about actions that allows us to account for aspects of the problem that are beyond the capabilities of other approaches, for instance, taking into account knowledge about the state of the world at various time points. Another advantage of using a general action theory is that the reasoner would be able to solve an activity recognition problem and then plan its own actions, possibly in response to what the observed agent is doing, using the same action domain description and observations, in the same formal framework. Some of the recent approaches to activity recognition are based on Hidden Markov Models, e.g. [Bui *et al.*, 2002; Geib and Goldman, 2005; Blaylock and Allen, 2006], and Probabilistic Grammars, e.g. [Pynadath and Wellman, 2000]. These approaches achieve high efficiency, but are limited in several respects. One major limitation is an inability to handle multiple activities occurring simultaneously. Other limitations of some of these approaches include an inability to utilize negative information, such as an observation that some action *did not occur*, observations about properties of the world, and a focus on answering a single type of query: what is the top-level activity being executed by the observed agent. In contrast, our approach based on a general theory of actions allows computing a more complete picture of the situation, that in addition to the top-level activities includes what the agent has done in the past, what is true of the world at each stage, what the agent may intend to do next, etc. There are also symbolic approaches, such as [Avrahami-Zilberbrand and Kaminka, 2005], which typically focus on matching observed actions to plans, achieving high efficiency at the cost of ignoring dynamic domain prop-

erties. An approach perhaps closer to ours is [Demolombe and Hamon, 2002]. As in our case, this approach also employs an action formalism, although a different one: the situation calculus. It differs in that it does not utilize notions of purposeful and justified actions, as we do, but relies on an explicit specification of which actions must *not occur* at particular points of an activity if it is to be recognized. This makes the definitions of activities more complicated and less elaboration tolerant. That approach is improved in [Goultiaeva and Lespeance, 2007] by reformalizing the problem in a way that allows incremental recognition of plans. Both of these approaches require the history to be a complete prefix of an activity in order for the recognition to work. They are thus not able to solve problems where the history is missing observations of earlier occurring actions, as in Example 3 below, for instance. These frameworks also require observations of actions to be made in the order the actions occur and cannot handle shared actions between activities. Our framework does not have these limitations. On the other hand, their approach allows more general forms of activities than the sequential ones we consider. A more general notion of activity is among our plans for future work.

It is worth pointing out as well that none of the above frameworks allows in the history statements of intention to execute an action or activity, since they do not employ an explicit notion of intended actions. The latter are important when employing activity recognition, for instance, for detecting errors (e.g. a patient that forgets to take some drugs) or for detecting attempts of deception.

Our main goal is a framework for activity recognition using a reasoning about actions language, together with an Answer Set Programming (ASP) [Gelfond and Lifschitz, 1990] formalization, that can be integrated with other reasoning modules [Baral and Gelfond, 2000], e.g. for planning and diagnosis. We chose the language $\mathcal{ALI}$ [Baral and Gelfond, 2005] because it belongs to a family of very general reasoning about actions formalisms with a solid theoretical foundation and also because this particular language already incorporates a notion of intended actions. An ASP formalization, in addition to a formal characterization, directly gives us correct implementations through solvers like Smodels [Niemelä and Simons, 1997]. Of course, solving a more general problem has a higher computational cost, so we cannot hope to achieve the execution times of some of the frameworks mentioned earlier. Nevertheless, the effectiveness of modern answer set solvers provides us with a competitive computational tool.

The rest of the paper is structured as follows: we start in Section 2 with an overview of a language for reasoning about intended actions. In Section 3, we explain the concepts of purposeful and justified activities and then formally characterize activity recognition as considered in this paper. This is then followed by our answer set programming formalization in Section 4. We conclude with some remarks in Section 5.

## 2 Reasoning about Intended Actions

The action description language $\mathcal{ALI}$ [Baral and Gelfond, 2005] extends similar action languages, in particular $\mathcal{AL}$ [Baral and Gelfond, 2000], with the capability of reasoning about intended actions. Next we give an overview of $\mathcal{ALI}$.

A domain description in $\mathcal{ALI}$ includes a set of *dynamic causal laws*, *static causal laws* and executability propositions. A set of such statements, called *action description*, describes a transition system which models how the world moves from one state to another as actions are executed. A separate set of constructs in the language is used to capture a *history*: a set of statements about observed values of fluents and occurrences of actions at specified time points. Given a domain description and a possibly incomplete history, the reasoning task is then to determine a complete trajectory that the world may have followed and that is compatible with the history. Additionally, the language $\mathcal{ALI}$ allows for reasoning about intended actions, thus it includes a construct for specifying in a history that at a given time the agent intends to execute a given action. The underlying principle is: *normally, unfulfilled intentions persist*, meaning that if the agent is not able to execute an intended action at a specified time (e.g. because the action was not executable at that time) then the intention persists until the agent successfully executes the action. The formal syntax and semantics of $\mathcal{ALI}$ follows.

A signature consists of two disjoint, finite sets: a set of elementary action names $A$, and a set of symbols $F$, called *fluents*, which represent properties of the domain that change when actions are executed. A *fluent literal* is a fluent $f$ or its negation, denoted by $\neg f$. A set of literals $Y$ is called *complete* if for every $f \in F$, $f \in Y$ or $\neg f \in Y$, and $Y$ is called *consistent* if there is no $f$ s.t. $f, \neg f \in Y$. A *state* is a complete and consistent set of fluent literals and represents one possible state of the domain. An *action* is a set $\{a_1, \ldots, a_n\}$ of elementary actions representing their simultaneous execution. Sequences of actions are lists of actions separated by commas and enclosed by $\langle \cdot \rangle$.

Given a signature $\Sigma = (A, F)$, a *transition diagram* over $\Sigma$ is defined as a directed graph $T$ where:

- the states of $T$ are the states of $\Sigma$, denoted by $\sigma_i$'s;
- the arcs of $T$ are labeled by actions of $\Sigma$.

A path $\langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$ of a transition diagram is called a *trajectory* of the domain.

As mentioned above, an action description consists of a set of statements understood as describing a transition diagram. These statements are of the following three forms ($a_e$ denotes an elementary action and $l_i$'s denote fluent literals):

- *dynamic causal laws*: $causes(a_e, l_0, [l_1, \ldots, l_n])$, stating that executing $a_e$ in a state where $l_1, \ldots, l_n$ hold causes $l_0$ to be true in the resulting state;
- *static causal laws*: $caused(l_0, [l_1, \ldots, l_n])$, stating that $l_0$ is caused to hold in every state where $l_1, \ldots, l_n$ hold;
- *executability propositions*: $impossible\_if(a_e, [l_1, \ldots l_n])$, stating that $a_e$ cannot be executed in a state where $l_1, \ldots, l_n$ hold.

The definition of the transition diagram specified by an action description requires the following notions and notation: An action $a$ is *executable* in a state $\sigma$ if there is no proposition $impossible\_if(a_e, [l_1, \ldots l_n])$ s.t. $a_e \in a$ and $\{l_1, \ldots, l_n\} \subseteq \sigma$. A set $S$ of fluent literals is *closed under*

a set $Z$ of static causal laws if $S$ includes the head $l_0$ of every static causal law s.t. $\{l_1, \ldots, l_n\} \subseteq S$. The set $Cn_Z(S)$ of *consequences* of $S$ under $Z$ is the smallest set of fluent literals that contains $S$ and is closed under $Z$. The notation $E(a_e, \sigma)$ is used to denote the set of all literals $l_0$ s.t. there is a dynamic causal law *causes*$(a_e, l_0, [l_1, \ldots, l_n])$ and $\{l_1, \ldots, l_n\} \subseteq \sigma$. Moreover, $E(a, \sigma) = \bigcup_{a_e \in a} E(a_e, \sigma)$.

An action description specifies a transition diagram that satisfies certain properties. One is that all the states of the transition diagram must satisfy the static causal laws. Second, if there is a transition from $\sigma$ to $\sigma'$ labeled by $a$, then $a$ must be executable in $\sigma$. Furthermore, $\sigma'$ must include the direct effects $E(a, \sigma)$ of $a$, the indirect effects that follow from the static causal laws and it must contain literals that are otherwise not affected by $a$ but are preserved by the common sense law of inertia. Formally, the transition system specified by an action description $AD$ is defined as follows.

**Definition 1** An action description $AD$ with signature $\Sigma$ describes the transition system $T = (S, R)$ where:

1. $S$ is the set of all the states of $\Sigma$ that are closed under the static causal laws of $AD$;
2. $R$ is the set of all triples $\langle \sigma, a, \sigma' \rangle$ s.t. $a$ is executable in $\sigma$ and $\sigma'$ is the fixpoint of the equation:
$$\sigma' = Cn_Z(E(a, \sigma) \cup (\sigma \cap \sigma'))$$
where $Z$ is the set of all the static causal laws in $AD$.

**Definition 2** A *history* is a set of propositions of the forms ($\alpha$ denotes an action sequence, $i$ a time point, and $l$ a fluent literal):

1. *intended*$(\alpha, i)$: action $\alpha$ is intended at time point $i$;
2. $(\neg)$*happened*$(a, i)$: $a$ (did not) happen at time point $i$;
3. *observed*$(l, i)$: $l$ was observed to hold at time point $i$.

The semantics of *happened* and *observed* is defined in the usual way for $\mathcal{A}$-like languages (see Definition 3 below). The semantics of *intended* is based on the following assumptions:

1. once an agent establishes the intention to execute an action, it does so as soon as the action is executable;
2. for an intended sequence $\langle a_1, \ldots, a_n \rangle$, $a_1$ is intended first and each $a_{i+1}$ in turn intended after $a_i$ executes;
3. if an intended action is not executable, the intention to execute it persists until it becomes possible to execute it.

A history is interpreted by trajectories of the background transition system. The following definition describes when a trajectory is a model of a history.

**Definition 3** Let $AD$ be an action description, $H$ a history, $P = \langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$ a trajectory and $1 \leq i \leq n$.

1. $P$ satisfies *observed*$(l, i)$ if $l$ is true in $\sigma_i$ ($l \in \sigma_i$). Similarly for initial state statements, $P$ satisfies *observed*$(l, 0)$ if $l$ is true in $\sigma_0$.
2. $P$ satisfies *happened*$(a, i)$ if $a \subseteq a_i$. In this case, $a$ is said to be *supported* at $i$. $P$ satisfies $\neg$*happened*$(a, i)$ if $a \not\subseteq a_i$.
3. $P$ satisfies *intended*$(a, i)$ if a) $a \subseteq a_i$; or b) $a$ is not executable in $\sigma_{i-1}$ (i.e. there is a proposition *impossible_if*$(a_e, \vec{l}) \in AD$ s.t. $a_e \in a$ and $\vec{l} \subseteq \sigma_{i-1}$), $i < n$ and $P$ satisfies *intended*$(a, i+1)$.
   If $a \subseteq a_i$, we say that $a$ *ends at* $i$+1 and that each element of $a$ is *supported at* $i$.
4. $P$ satisfies *intended*$(\alpha, i)$ where $\alpha = \langle a'_1, \ldots, a'_m \rangle$ and $m > 1$, if $P$ satisfies *intended*$(a'_1, i)$ and $P$ satisfies *intended*$(\langle a'_2, \ldots, a'_m \rangle, j)$ where $a'_1$ ends at $j$ in $P$. We say that $\alpha$ *ends at* $k$ in $P$, if $a'_m$ ends at $k$ in $P$.
5. $P$ is a model of a history $H$ if it satisfies all the statements of $H$ and, for all $1 \leq i \leq n$, all the elements of $a_i$ are supported at $i$ in $P$.

# 3 Activity Recognition

Reasoning about intended actions in the language $\mathcal{ALI}$ is done in terms of the trajectories specified by the history. Our goal here is to characterize activity recognition similarly in terms of models of the recorded history. The differing nature of the activity recognition problem requires some elaboration of the framework. For once, intuitively activity recognition is done from the perspective of an external observer of the agent that is executing the actions. Also, in addition to the background action description and the history, in activity recognition the reasoner typically has additional knowledge in the form of a set of activities that the agent being observed may do and sometimes information about which actions are "purposeful." Let us elaborate these two points a bit further.

## 3.1 Named Activities

Our approach to activity recognition is based on the availability of a set of background activities that the observed agent may do. These serve as hypothesis space to the recognition system. In our case these activities will be represented as pairs $(s, \alpha)$ where $s$ is the name of the activity and $\alpha$ a sequence of actions (including other activities). Thus we extend the signature of the language with an additional set $C$ of activity names. An action description will contain a set of pairs $(s, \alpha)$ with one pair for each $s \in C$. We will often use $s$ to refer to activity $(s, \alpha)$. For the sake of simplicity, sequences $\alpha$ in named activities are assumed not to repeat actions.

Two examples from the cooking domain are:

$$(ccm, \langle mk\_marinara, mix\_chicken\_marinara \rangle)$$

$$(cfm, \langle mk\_fettuccini, mk\_marinara, mix\_fettuccini\_marinara \rangle)$$

We assume that only actions can be observed to occur and thus will not use activity names in *happened* statements. On the other hand, we do allow activity names in *intended* statements as part of the history, as we consider the possibility that the observed agent declares its intentions or that the activity recognizer is otherwise informed of those intentions.

## 3.2 Purposeful Actions

By purposeful actions we mean actions whose execution in isolation, as opposed to as part of a more complex activity, is considered reasonable. For example, one may consider the action of taking a bus as not purposeful since normally a person does not take a bus for the sake of taking a bus, but does so as part of a more complex activity such as commuting to work. Here we treat purposefulness as a fixed (non fluent) property of activities. A more elaborate treatment of purposefulness intuitively seems to require this notion to be context

dependent and to be captured as a default. We plan to consider more general notions of purposefulness in future work.

Purposeful actions are simply declared to be so by means of statements of the form *purposeful*$(c)$, where $c$ is an action or an activity. Actions not declared to be purposeful are assumed not to be purposeful. The next section describes how this knowledge influences reasoning about intended actions.

### 3.3 Formal Characterization

In our formalization of activity recognition we do not allow a history to state that a named activity happened, only elementary actions can be observed. We moreover assume that all the observed actions were intentional.

The definition of satisfaction of history statements by a trajectory is given in Def. 3. While activities cannot be observed to happen, we allow that the reasoner may be informed that the observed agent intends to execute a named activity. This means that we must extend histories by allowing statements *intended*$(s, i)$ where $s$ is an activity name, and extend the definition of satisfaction of such statements by a trajectory $P$.

**Definition 4** For a named activity $(s, \alpha)$, a trajectory $P$ satisfies a statement *intended*$(s, i)$ if $P$ satisfies *intended*$(\alpha, i)$.

For simplicity we will assume that sequences $\alpha$ relevant to a history are given a name $s$ and that *intended*$(s, i)$ is used instead of *intended*$(\alpha, i)$. We will also overload *actions* to include named activities and actions as defined earlier.

Before we introduce models of a history, we need some terminology. We say that an action (including named activities) $c$ *starts at* $i$ in a trajectory $P$ if $P$ satisfies *intended*$(c, i)$ but does not satisfy *intended*$(c, i-1)$, i.e. it is said to start when it becomes intended. A named activity $(s, \langle a_1, \ldots, a_n\rangle)$ *ends at* $i$ in $P$ if $a_n$ ends at $i$ in $P$ (as in Def. 3).[1] Furthermore, an action $c$ is said to be *in progress* at $k$ in $P$ if $c$ starts at $i$ and ends at $j$ in $P$ and $i \leq k < j$. Henceforth we will omit $P$ when clear from context and say $c$ starts at $i$, $c$ ends at $j$, etc.

Next we define the key notion of *justified actions*. This notion captures the intuition that if an action that is not purposeful is believed to occur or be intended at $i$, then it must be part of an activity in progress at the same time $i$.

**Definition 5** Let $AD$ be an action description, $P$ be a trajectory and $c$ be an action.

1. $c$ is *justified by* $c$ at $i$ if *purposeful*$(c) \in AD$;
2. $c$ is *justified by* $s$ at $i$ if
   (a) $(s, \alpha)$ is a named activity in $AD$,
   (b) $c$ appears in $\alpha$,
   (c) $s$ is in progress at $i$,
   (d) $s$ does not justify $c$ at an earlier time point in its current execution, that is, if $l$ is the latest start time of $s$ such that $l < i$, then $s$ does not justify $c$ at $k$ such that $l \leq k < i$.

We say $c$ is *justified* at $i$ if $c$ is justified by $b$ at $i$ for some $b$.

We are now ready to define models of a history. This definition must take into account whether actions are justified or not for the purpose of reasoning about activity recognition.

[1] "Starts" and "ends" mark the period between an action first becoming intended and the termination of its execution.

In addition to satisfying the history, a trajectory must satisfy a number of additional conditions. Condition (2) below precludes vacuous actions from models. Condition (3) intuitively says that for every action in progress there must be at least one action that justifies it from start to end. Condition (4) says that an activity cannot end if an action that appears in its sequence is still intended, unless that action is justified by some other activity. Finally, Condition (5) says that at the end of the trajectory, no intended actions remain.

**Definition 6** A trajectory $P = \langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n\rangle$ is a model of a history $H$ of an action description $AD$ if the following conditions hold:

1. $P$ satisfies all the statements of $H$;
2. for each $1 \leq i \leq n$, all elements of $a_i$ are supported at $i$;
3. for every action $c$ such that $P$ satisfies *intended*$(c, i)$ and $c$ starts at $i$ and ends at $j$, there is an action $c'$ such that $c$ is justified by $c'$ at $k$ for every $i \leq k < j$;
4. for every activity $(s, \langle c_1, \ldots, c_m\rangle)$ s.t. $s$ ends at $i$+1, there is no action $c_k$, $1 \leq k < m$, in the sequence of $s$ s.t.
   (a) $P$ satisfies *intended*$(c_k, i)$,
   (b) $c_k$ is justified by $s$ at $i$,
   (c) there is no $s' \neq s$ such that $c_k$ is justified by $s'$ at $i$;
5. for every action $c$, if $c$ is in progress at $n$, $c$ ends at $n$+1.

**Example 2** Consider again the cooking domain. Suppose that we have a history containing the following statements: *intended*(*mk_fettuccini*, 1),*intended*(*mix_chicken_marinara*, 3). Assuming no concurrency, this history has no models of length less than 4. It has one model of length 4 with actions: *mk_fettuccini*, *mk_marinara*, *mix_chicken_marinara*, *mix_fettuccini_marinara*, occurring in that order. Intuitively, two activities are occurring: cook chicken marinara (*ccm*), in progress from time 2 to 3, and cook fettuccini marinara (*cfm*), in progress from 1 to 4. They share the action *mk_marinara*, which is justified by both *ccm* and *cfm* at time 2.

It has 4 models of length 5. One of which contains the actions *mk_marinara*, *mk_fettuccini*, *mix_chicken_marinara*, *mk_marinara*, *mix_fettuccini_marinara*. The same *ccm* and *cfm* are occurring, with *ccm* in progress from 1 to 3 and *cfm* from 1 to 5. In this case *mk_marinara* is not shared as it occurs for *ccm* before it becomes intended for *cfm*. In the other two models, *ccm* and 'cook fettuccini alfredo' (*cfa*) occur. In one model *mk_marinara* occurs at 1 and *mk_fettuccini* at 2. In the other they occur in the opposite order.

## 4 Formalization in ASP

For lack of space we must rely on familiarity with ASP encodings of dynamic domains (e.g. [Baral, 2003]) which are a component of our formalization. We only describe the main components required on top of the transition system encoding. In the rules shown below, domain predicates should be used in the usual way making them safe. We do not show them to save space. In our experiments we use the Smodels construct `#domain` to specify domains for all the variables that appear in the rules. We start with the component that, given a statement *intended*$(c, i)$ in the history, conjectures that some activity $s$ containing $c$ is in progress.

```
inprogress(S,I):- component(C,K,S),
                  intended(C,I), K <= I,
                  not other_justified(C,I,S).

other_justified(C,I,S):- component(C,K,S),
                         justified(C,I,S1),
                         neq(S,S1).
```

From conjecturing that an activity is in progress one can conclude that it was intended at some point:

```
intended(S,I):- inprogress(S,I),
                not inprogress(S,I-1).
intended(S,I):- inprogress(S,I), ends(S,I).
```

Next we describe the component that captures the notion of justified actions, starting with self-justified actions:

```
justified(C,I,C):- inprogress(C,I),
                   purposeful(C).
```

The following rule captures item 2 of Definition 5:

```
justified(C,I,S):- inprogress(C,I),
                   component(C,K,S),
                   inprogress(S,I),
                   not justified_before(C,S,I).
```

The above rules conjecture activities to justify actions. One of the main components of the formalization "propagates" this knowledge across time. This component has two parts with respect to a time point $i$: one for inference about what holds at times preceding $i$ and the other for times later than $i$. We start with the rules for reasoning about later time points.

The following two rules encode directly the definition of *in progress*:

```
inprogress(S,I) :- starts(S,I).

inprogress(S,I) :- inprogress(S,I-1),
                   not ends(S,I).
```

The components of an activity become intended in sequence:

```
intended(C,I):- starts(S,I),
                component(C,1,S).

intended(C2,I):- inprogress(S,I),
                 component(C2,K,S),
                 component(C1,K-1,S),
                 ends(C1,I),
                 justified(C1,I-1,S).
```

The component for inference at earlier time points includes a rule saying that if an activity is in progress and it justifies one of its components that is not the first, then it must have been in progress in the previous time point:

```
inprogress(S,I):- inprogress(S,I+1),
                  intended(A,I+1),
                  component(A,K,S), K > 1,
                  justified(A,I+1,S).
```

From inferred action occurrences, it is possible to further infer intentions of the agent before the action was executed, especially if the action is not self-justified. The following rules say, roughly, that if the action is a component of an activity and the activity has an earlier component, then either the earlier component is intended in the preceding time point or the action that occurred was intended in the preceding time point.

```
intended(A1,I):- inprogress(S,I+1),
                 occurs(A2,I+1),
                 component(A2,K,S),
                 component(A1,K-1,S),
                 justified(A2,I+1,S),
                 not -occurs(A1,I),
                 not intended(A2,I).

intended(A2,I):- inprogress(S,I+1),
                 occurs(A2,I+1),
                 component(A2,K,S),
                 component(A1,K-1,S),
                 justified(A2,I+1,S),
                 not intended(A1,I).
```

Finally, the following rules capture conditions (3–5) in Def. 6:
Condition (3):

```
:- inter(C,I,I1), not full_just(C,I,I1).

full_just(C,I,I1):- full_justified(C,C1,I,I1).

full_justified(C,C1,I,I1):- justified(C,I,C1).

full_justified(C,C1,I,I1):-
                  justified(C,I,C1), I < I1,
                  full_justified(C,C1,I+1,I1).
```

Condition (4):

```
:- ends(S,I+1), intended(C,I),
   justified(C,I,S), not other_justified(C,I,S),
   length(S,K), not component(C,K,S).
```

Condition (5):

```
:- intended(C,n).
```

In the last rule, $n$ is a constant defined to be the maximum length of the trajectories to be considered, as typically done in ASP encodings of transition systems. In this case, however, it is not really necessary to find a minimal $n$ to find solutions, since all actions must be justified. The only drawback of using too large an $n$ is that the size of the grounded program increases.

The above set of rules, plus a few more omitted for space reasons, is denoted by $\Pi_{ar}$. A history $H$ is encoded directly as a set of facts and will be denoted by $\pi(H)$. The translation of a domain description $AD$ is denoted by $\pi(AD)$. For each named activity $(s, \langle c_1, \ldots, c_m \rangle)$ $\pi(AD)$ includes the facts:

```
activity(s). length(s,m).
component(c1,1,s). ... component(cm,m,s).
```

It also includes a fact `purposeful(c)` for each purposeful action or named activity $c$. Dynamic causal laws, static causal laws and executability propositions describe a transition system that is encoded in the usual way for similar action languages. We omit those rules here.

One of the interesting aspects of this formalization is the reasoning about state properties that is captured. The reasoner may infer the values of fluents from intentions and action occurrences, for instance, that a precondition $p$ of an action $a$ is false from the observation that $a$ was intended but did not occur. This is partly enabled by means of stating that in the initial state any fluent for which there is no information may be assumed true or it may be assumed false, as long as consistency is preserved. This is captured by a pair of rules for each fluent $f$:

```
holds(f,1):- not holds(-f,1).
holds(-f,1):- not holds(f,1).
```

which results in multiple answer sets corresponding to the various alternatives.
Intention of atomic actions is captured by a set $\Pi_I$ including:

```
intended(A,I):- happened(A,I).

occurs(A,I):- intended(A,I), not -occurs(A,I).

intended(A,I+1):- intended(A,I), -occurs(A,I),
                  not -inteded(A,I+1).

inprogress(A,I):- starts(A,I).

inprogress(A,I):- inprogress(A,I-1),
                  not ends(A,I).
```

The complete formalization of an activity recognition problem is given by $\pi(AD, H) = \Pi_I \cup \Pi_{ar} \cup \pi(AD) \cup \pi(H)$. The models $\pi(AD, H)$ induce trajectories as follows.

**Definition 7** Let $A$ be a subset of the literals of a given program $\pi(AD, H)$. $A$ is said to define the trajectory $\langle \sigma_0, a_1, \sigma_1, \ldots, \sigma_n, a_n \rangle$ if $\sigma_i = \{l \mid \mathit{holds}(l, i) \in A\}$ and $\mathit{occurs}(a_j, j) \in A$ for all $0 \leq i \leq n$ and $1 \leq j \leq n$

**Theorem 1** For an action description $AD$ and history $H$, a trajectory $P$ without concurrency (i.e. all actions are singletons or empty sets) is a model of $H$ iff $P$ is defined by an answer set of the program $\pi(AD, H)$.

**Example 3** For the Cooking domain, using the facts:

```
intended(make_fettuccini,1).
happened(mix_chicken_marinara,3).
```

and max trajectory length of 4 yields an answer set with:

```
inprogress(cfm,1)
inprogress(cfm,2)  inprogress(ccm,2)
inprogress(cfm,3)  inprogress(ccm,3)
inprogress(cfm,4)
justified(make_fettuccini,1,cfm)
justified(make_marinara,2,ccm)
justified(make_marinara,2,cfm)
justified(mix_chicken_marinara,3,ccm)
justified(mix_fettuccini_marinara,3,cfm)
justified(mix_fettuccini_marinara,4,cfm)
occurs(make_fettuccini,1)
occurs(make_marinara,2)
occurs(mix_chicken_marinara,3)
occurs(mix_fettuccini_marinara,4)
```

## 5 Conclusion

We have introduced a new approach to activity recognition based on a formal theory of actions and a notion of intended actions. Our approach is based on using knowledge about the intention and the occurrences of non-purposeful actions to conjecture that more complex purposeful activities may be occurring. In addition to $\mathcal{A}$-type action language domain descriptions with a transition system-based semantics, we provide a formalization in ASP. Some advantages of our approach are a strong temporal reasoning component that allows taking into account observations about the dynamic properties of the world in addition to observations about action occurrences. It is also allows simultaneous activities, shared actions, use of observations about non-occurrence of actions, and explicit statements about intention to execute actions. Consequently, the set of queries that can be answered is also much larger than in other approaches.

There are a number of directions the framework can be extended: adding probabilities using e.g. a language like P-log [Baral *et al.*, 2004]; recognition of failed actions and the persistence of the intention to execute those; recognition that an agent abandoned an activity in the middle of the execution. We plan to look at this in future work.

## References


[Avrahami-Zilberbrand and Kaminka, 2005] D. Avrahami-Zilberbrand and G.A. Kaminka. Fast and complete symbolic plan recognition. In *IJCAI*, 2005.

[Baral and Gelfond, 2000] C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In J. Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer, 2000.

[Baral and Gelfond, 2005] C. Baral and M. Gelfond. Reasoning about intended actions. In *AAAI*, 2005.

[Baral *et al.*, 2004] C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. *LPNMR*, 2004.

[Baral, 2003] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[Blaylock and Allen, 2006] N. Blaylock and J. Allen. Fast hierarchical goal schema recognition. In *AAAI*, 2006.

[Bui *et al.*, 2002] H.H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 2002.

[Demolombe and Hamon, 2002] R. Demolombe and E. Hamon. What does it mean that an agent is performing a typical procedure? In *AAMAS*, 2002.

[Geib and Goldman, 2005] C.W. Geib and R.P. Goldman. Partial observability and probabilistic plan/goal recognition. In *MOO Workshop*, 2005.

[Gelfond and Lifschitz, 1990] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *ICLP*, 1990.

[Goultiaeva and Lespeance, 2007] A. Goultiaeva and Y. Lespeance. Incremental plan recognition in an agent programming framework. In *PAIR Workshop*, 2007.

[Kautz and Allen, 1986] H. Kautz and J. Allen. Generalized plan recognition. In *AAAI*, 1986.

[Niemelä and Simons, 1997] I. Niemelä and P. Simons. Smodels–an implementation of the stable models and well-founded semantics for normal logic programs. *LPNMR'97*.

[Pynadath and Wellman, 2000] D.V. Pynadath and M.P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *UAI*, 2000.